CSE 5525 Homework #1: Sentiment Classification

Deadline: **11:59PM on 09/08/2021**.

Academic Integrity

You can discuss the homework assignments with other students and study course materials together towards solutions. However, all of the code and report you write must be your own!!! Plagiarism will be automatically detected on Carmen (by comparing previous and current HWs) as well as manually checked by TA.

If the instructor or TA suspects that a student has committed academic misconduct in this course, they are obligated by university rules to report their suspicions to the Committee on Academic Misconduct. Please see more in Statements for CSE5525 on Carmen pages.

Goals

The main goal of Assignment #1 is for you to get familiar with extracting features and training classifiers on text data. You'll get a sense of what the standard machine learning workflow looks like (reading in data, training, and testing), how standard learning algorithms work, and how the feature design process goes.

Timeline & Credit

You will have around 2 weeks to work on this programming assignment. We currently use a 100-point scale for this homework, but it will take 10% of your final grade.

Questions?

Please post on Microsoft Teams to get timely help from other students, the TA, and the instructor. Remember that participation takes 5% of your final grade. You can show your participation by actively answering others' questions! **Besides, everyone can benefit from checking what has been asked previously.** So, please try to avoid directly sending emails to the instructor/TA. Thanks!

1 Dataset and Code

1.1 Data

You'll be using the movie review dataset in Socher et al. [1]. This is a dataset of movie review snippets taken from Rotten Tomatoes. We are tackling a simplified version of this task which frequently appears in the literature: positive/negative binary sentiment classification of sentences, with neutral sentences discarded from the dataset. The data files given to you contain of newline-separated sentiment examples, consisting of a label (0 or 1) followed by a tab, followed by the sentence, **which has been tokenized but not lowercased**. The data has been split into a train, development (dev), and blind test set. On the blind test set, you do not see the labels and only the sentences are given to you. The framework code reads these in for you.

1.2 Getting started

Download the code and data. Expand the file and change into the directory. To confirm everything is working properly, run:

```
python sentiment_classifier.py --model TRIVIAL --no_run_on_test
```

This loads the data, instantiates a TrivialSentimentClassifier that always returns 1 (positive), and evaluates it on the training and dev sets. The reported dev accuracy should be Accuracy: 444 / 872 = 0.509174. Always predicting positive isn't so good!

1.3 Framework code.

You have been given the framework code to start with:

`sentiment_classifier.py` is the main class. **Do not modify this file for your final submission** (you can do whatever you need before submission, though). The main method loads in the data, initializes the feature extractor, trains the model, and evaluates it on train, dev, and blind test, and writes the blind test results to a file. It uses `argparse` to read in several command line arguments. You should generally not need to modify the paths. –model and –feats control the model specification. This file also contains evaluation code.

`models.py` is the primary file you'll be modifying. It defines base classes for the `FeatureExtractor` and the classifiers, and defines `train_logistic_regression`[1], which you will be implementing. `train_model` is your entry point which you may modify if needed.

2 What You Need to Do

2.1 Part 1: Feature extraction (20 points)

To build a classifier, you will first need a way of mapping from each sentence (lists of strings) to a feature vector, a process called feature extraction or featurization.

Q1 (10 points): Implement `UnigramFeatureExtractor` to extract unigram bag-of-words features. A unigram feature vector will be a sparse vector with length equal to the vocabulary size. There is no one right way to define unigram features. For example, do you want to throw out low-count words? Do you want to lowercase? Do you want to discard stopwords? Do you want to clip counts to 1 or count all occurrences of each word? In practice, you can try different ways and see which way helps a classifier achieve a better performance on dev set. In your report, briefly describe what you did and what feature values mean.

You can use the provided `Indexer` class in `utils.py` to map from string-valued feature names to indices. Note that later when you have other types of features in the mix (e.g., bigrams), you can still get away with just using a single Indexer: you can encode your features with "magic words" like Unigram=great and Bigram=great—movie. This is a good strategy for managing complex feature sets.

Q2 (10 points): Implement `BigramFeatureExtractor` to extracting bigram bag-of-words features. Bigrams are adjacent pairs of words in the text (e.g., `Bigram=great|movie` while `Unigram=great`). Implement `BigramFeatureExtractor`, similarly as `UnigramFeatureExtractor` above. Briefly describe what you did and what feature values mean in your report.

Note that: **Feature vectors.** Since there are a large number of possible features, it is always preferable to represent feature vectors sparsely. That is, if you are using unigram features with a 10,000 word vocabulary, you should not be instantiating a 10,000-dimensional vector for each example, as this is very inefficient. Instead, you want to maintain a list of only the nonzero features and their counts. You might find `Counter` from the `collections` package useful for storing sparse vectors like this.

Weight vector. The most efficient way to store the weight vector is a fixed-size numpy array.

2.2 Part 2: Implementing Logistic Regression Classifier (80 points)

In this part, you'll implement a logistic regression classifier and test its performance using the unigram bag-of-words feature set and the bigram bag-of-words feature set (extracted in the previous part).

[1]Note that the code given to you also defines the `train_perceptron` (NOT REQUIRED) method, which is out of the scope of this homework and you don't need to implement.

Q3 (50 points) Implement logistic regression and show its performance with unigram features. You need to implement logistic regression training in `train_logistic_regression` and `LogisticRegressionClassifier` in `models.py`. Note that you have a lot of freedom to modify `models.py` in your implementation; however, you are expected to show how you computed the loss function, the gradients, and the gradient descent algorithm (e.g., at least the simplest version of Algorithm 5 in the Eisenstein textbook[2]) to minimize the loss function. Do NOT directly call existing libraries (such as `sklearn.linear_model.LogisticRegression`) or other well-implemented logistic regression classifiers.

Report your model's performance on the training/dev dataset using the unigram bag-of-words feature set. While we don't require you to obtain a certain accuracy for full credit, for your reference, in this setting you should be able to easily get at least 70% accuracy on the dev set and your code should finish running in dozens of (e.g., less than 30) seconds.

Q4 (10 points) Logistic regression + bigram features. Report your implemented logistic regression model's performance using the bigram bag-of-words feature set, on the training/dev dataset.

Q5 (10 points): Implement `BetterFeatureExtractor` to explore better features. Things you might try: combining unigrams together with bigrams, other types of n-grams, tf-idf weighting, keeping/clipping your word frequencies, discarding rare words, discarding stopwords, etc. Report your implemented logistic regression model's performance using your designed feature set, on the training/dev dataset. Briefly describe what you tried in your report.

Q6 (10 points) Training analysis. Which feature setting (unigram, bigram, or the one you implemented in `BetterFeatureExtractor`) performs best on the dev set? Under the best feature setting, plot (using matplotlib or another tool) the training loss and development accuracy (i.e., accuracy on dev set) of logistic regression vs. number of training iterations. Plot the two curves by choosing 3 different step sizes (such as [0.01, 0.1, 1]; you can decide your own). What do you observe?

3 What You Should Submit and Our Expectation

Submission. You should submit the following files to Carmen **as three separate file uploads** (not a zip file):

1. A PDF file of your answers to questions Q1-Q6 as your report.
2. Blind test set output in a file named `test-blind.output.txt`. The code produces this by default, but make sure you include the right version (e.g., the results of your model with the best feature setting)!
3. `models.py`, which should be submitted as an individual file upload. **Do not modify or upload `sentiment_classifier.py`, `sentiment_data.py`, or `utils.py`**. Please put all of your code in `models.py`.

Expectation. Beyond your writeup, your submission will be evaluated on several axes:

1. Execution: your code should train and evaluate within a reasonable amount of time (e.g., dozens of seconds; usually less than 30s) without crashing.
2. Reasonable accuracy on the development set of your logistic regression classifier under each feature setting. You should be able to get at least 70% accuracy, and **please conduct error analysis if not**.
3. Reasonable accuracy on the blind test set. You should run prediction with your best feature setting (which is determined based on the dev set performance; the setting you used for Q6) and submit the test output file. Our TA will compute the accuracy on this blind test set.

Before you submit, make sure that the following commands work:

```
python sentiment_classifier.py --model LR --feats UNIGRAM
python sentiment_classifier.py --model LR --feats BIGRAM
python sentiment_classifier.py --model LR --feats BETTER
```

These commands should all print dev results and write blind test output to the file by default (so, **make sure you have the right version submitted**).

[2] https://github.com/jacobeisenstein/gt-nlp-class/blob/master/notes/eisenstein-nlp-notes.pdf

3.1 Acknowledgment

This homework assignment is largely adapted from NLP courses by Dr. Greg Durrett at UT Austin.

References

[1] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.